Available online at www.sciencedirect.com
ScienceDirect
Discrete Applied Mathematics 157 (2009) 2143–2152
DISCRETE APPLIED MATHEMATICS
www.elsevier.com/locate/dam

On some algorithmic problems regarding the hairpin completion

Florin Manea[a], Carlos Martín-Vide[b], Victor Mitrana[a,b,*]

[a] *Faculty of Mathematic and Computer Science, University of Bucharest, Str. Academiei 14, 010014, Bucharest, Romania*
[b] *Research Group in Mathematical Linguistics, Rovira i Virgili University, Pl. Imperial Tarraco 1, 43005, Tarragona, Spain*

Received 9 October 2006; received in revised form 17 June 2007; accepted 10 September 2007
Available online 20 February 2008

Abstract

We consider a few algorithmic problems regarding the hairpin completion. The first problem we consider is the *membership problem* of the hairpin and iterated hairpin completion of a language. We propose an $\mathcal{O}(nf(n))$ and $\mathcal{O}(n^2 f(n))$ time recognition algorithm for the hairpin completion and iterated hairpin completion, respectively, of a language recognizable in $\mathcal{O}(f(n))$ time. We show that the n factor is not needed in the case of hairpin completion of regular and context-free languages. The n^2 factor is not needed in the case of iterated hairpin completion of context-free languages, but it is reduced to n in the case of iterated hairpin completion of regular languages. We then define the *hairpin completion distance* between two words and present a cubic time algorithm for computing this distance. A linear time algorithm for deciding whether or not the hairpin completion distance with respect to a given word is connected is also discussed. Finally, we give a short list of open problems which appear attractive to us.

Keywords: Hairpin completion; Iterated hairpin completion; Membership problem; Hairpin completion distance

1. Introduction

A DNA molecule consists of a double strand, each DNA single strand being composed by nucleotides which differ from each other by their bases: A (adenine), G (guanine), C (cytosine), and T (thymine). The two strands which form the DNA molecule are kept together by the hydrogen bond between the bases: A always bonds with T, while C bonds with G. This paradigm is usually referred as the *Watson–Crick complementarity*. Another important biological principle is that of *annealing*, which refers to fusing two single stranded molecules by a complementary base. This operation of fusing two single stranded molecules by a complementary base requires a heated solution containing the two strands which is cooled down slowly. It is known that a single stranded DNA molecule might produce a hairpin structure, a phenomenon based on these two biological principles. In many DNA-based algorithms, these DNA molecules cannot be used in the subsequent computations. Hairpin or hairpin-free DNA structures have numerous applications to DNA computing and molecular genetics. In a series of papers (see, e.g., [4–6]) the problem of finding sets of DNA sequences which are unlikely to lead to "bad" hybridizations is considered. On the other hand, these molecules which may form a hairpin structure have been used as the basic feature of a new computational model

* Corresponding author at: Faculty of Mathematic and Computer Science, University of Bucharest, Str. Academiei 14, 010014, Bucharest, Romania. Fax: +40 21 315 69 90.
E-mail addresses: flmanea@gmail.com (F. Manea), carlos.martin@urv.cat (C. Martín-Vide), mitrana@fmi.unibuc.ro (V. Mitrana).

0166-218X/$ - see front matter
doi:10.1016/j.dam.2007.09.022

reported in [11], where an instance of the 3-SAT problem has been solved by a DNA-algorithm in which the second phase is mainly based on the elimination of hairpin structured molecules. Different types of hairpin and hairpin-free languages are defined in [9,2], and more recently in [8], where they are studied from a language theoretical point of view.

The source of inspiration for introducing in [3], following an idea from [1], a new formal operation on words, namely *hairpin completion*, consists of three biological principles. Besides the Watson–Crick complementarity and annealing, the third biological phenomenon is that of *lengthening DNA by polymerases*. This phenomenon produces a complete double stranded DNA molecule as follows: one starts with two single strands such that one (usually called *primer*) is bonded to a part of the other (usually called *template*) by Watson–Crick complementarity and a *polymerization buffer* with many copies of the four nucleotides. Then polymerases will concatenate to the primer by complementing the template.

In this note we consider a few algorithmic problems regarding the hairpin completion. The first problem we consider is the *membership problem* of the hairpin and iterated hairpin completion of a language. We propose an $\mathcal{O}(nf(n))$ and $\mathcal{O}(n^2 f(n))$ time recognition algorithm for the hairpin completion and iterated hairpin completion, respectively, of a language recognizable in $\mathcal{O}(f(n))$ time. We show that the n factor is not needed in the case of hairpin completion of regular and context-free languages. The n^2 factor is not needed in the case of iterated hairpin completion of context-free languages, but it is reduced to n in the case of iterated hairpin completion of regular languages. We then define the *hairpin completion distance* between two words as the minimal number of hairpin completions applied to one word in order to get the other. A cubic time algorithm for computing this distance is presented. Given a word x we say that the hairpin completion distance is *connected* with respect to x if for every integer $n \geq 1$ there exists a word y_n such that the hairpin completion distance between x and y_n is exactly n. We discuss a linear time algorithm for deciding whether or not the hairpin completion distance with respect to a given word is connected.

The paper is structured as follows. Section 2 presents the basic concepts, notions and notations. Section 3 is devoted to the membership problem and Section 4 deals with the hairpin completion distance. We conclude with a short list of open problems which seems mathematically attractive to us.

2. Basic definitions

We assume the reader to be familiar with the fundamental concepts of formal language theory and automata theory, particularly the notions of grammar and finite automaton [10].

An alphabet is always a finite set of letters. For a finite set A we denote by $\text{card}(A)$ the cardinality of A. The set of all words over an alphabet V is denoted by V^*. The empty word is written λ; moreover, $V^+ = V^* \setminus \{\lambda\}$. Given a word w over an alphabet V, we denote by $|w|$ its length, while $|w|_a$ denotes the number of occurrences of the letter a in w. If $w = xyz$ for some $x, y, z \in V^*$, then x, y, z are called prefix, subword, suffix, respectively, of w. For a word w, $w[i..j]$ denotes the subword of w starting at position i and ending at position j, $1 \leq i \leq j \leq |w|$. If $i = j$, then $w[i..j]$ is the i-th letter of w which is simply denoted by $w[i]$.

Let Ω be a "superalphabet", that is an infinite set such that any alphabet considered in this paper is a subset of Ω. In other words, Ω is the *universe* of the languages in this paper, i.e., all words and languages are over alphabets that are subsets of Ω. An *involution* over a set S is a bijective mapping $\sigma : S \longrightarrow S$ such that $\sigma = \sigma^{-1}$. Any involution σ on Ω such that $\sigma(a) \neq a$ for all $a \in \Omega$ is said here to be a *Watson–Crick involution*. Despite this being nothing more than a fixed point-free involution, we prefer this terminology since the hairpin completion defined later is inspired by the DNA lengthening by polymerases, where the Watson–Crick complementarity plays an important role. Let $\bar{\cdot}$ be a Watson–Crick involution fixed for the rest of the paper. The Watson–Crick involution is extended to a morphism from Ω^* to Ω^* in the usual way. We say that the letters a and $\overline{a}$ are complementary to each other. For an alphabet V, we set $\overline{V} = \{\overline{a} \mid a \in V\}$. Note that V and $\overline{V}$ can intersect and they can be, but need not be, equal. Remember that the DNA alphabet consists of four letters, $V_{\text{DNA}} = \{A, C, G, T\}$, which are abbreviations for the four nucleotides, and we may set $\overline{A} = T, \overline{C} = G$.

We denote by $(\cdot)^R$ the mapping defined by ${}^R : V^* \longrightarrow V^*$, $(a_1 a_2 \dots a_n)^R = a_n \dots a_2 a_1$. Note that ^{R} is an involution and an *anti-morphism* ($(xy)^R = y^R x^R$ for all $x, y \in V^*$). Note also that the two mappings $\bar{\cdot}$ and $\cdot^R$ commute; namely, for any word x, $(\overline{x})^R = \overline{x^R}$ holds.

Let V be an alphabet; for any $w \in V^+$ we define the *k-hairpin completion* of w, denoted by $\rightarrow_k$, for some $k \geq 1$, as follows:

$$w \rightharpoonup_k = \{\overline{\gamma^R} w | w = \alpha\beta\overline{\alpha^R}\gamma, |\alpha| = k, \alpha, \beta \in V^+, \gamma \in V^*\}$$
$$w \rightharpoondown_k = \{w\overline{\gamma^R} | w = \gamma\alpha\beta\overline{\alpha^R}, |\alpha| = k, \alpha, \beta \in V^+, \gamma \in V^*\}$$
$$w \rightarrow_k = w \rightharpoonup_k \cup w \rightharpoondown_k .$$

The *hairpin completion* of w is defined by

$$w \rightarrow = \bigcup_{k \geq 1} w \rightarrow_k .$$

Clearly, $w \rightarrow_{k+1} \subseteq w \rightarrow_k$ for any $w \in V^+$ and $k \geq 1$, hence $w \rightarrow = w \rightarrow_1$. The hairpin completion operation is naturally extended to languages by

$$L \rightarrow_k = \bigcup_{w \in L} w \rightarrow_k \qquad L \rightarrow = \bigcup_{w \in L} w \rightarrow .$$

The iterated version of the hairpin completions is defined as usual by

$$w(\rightarrow_k)^0 = \{w\}, \qquad w(\rightarrow_k)^{n+1} = (w(\rightarrow_k)^n) \rightarrow_k, \qquad w(\rightarrow_k)^* = \bigcup_{n \geq 0} w(\rightarrow_k)^n$$
$$w(\rightarrow)^0 = \{w\}, \qquad w(\rightarrow)^{n+1} = (w(\rightarrow)^n) \rightarrow, \qquad w(\rightarrow)^* = \bigcup_{n \geq 0} w(\rightarrow)^n$$
$$L(\rightarrow_k)^* = \bigcup_{w \in L} w(\rightarrow_k)^* \qquad L(\rightarrow)^* = \bigcup_{w \in L} w(\rightarrow)^*.$$

3. Membership problem

In this section we consider the complexity of the membership problem for the hairpin completion and iterated hairpin completion of a language.

3.1. The case of non-iterated hairpin completion

In [3] one proves that the class of polynomially recognizable languages **P** is closed under non-iterated hairpin completion. More precisely, one proves:

Proposition 1 ([3]). *If the membership problem for a given language L is decidable in $\mathcal{O}(f(n))$, then the membership problem for $L \rightarrow_k$ is decidable in $\mathcal{O}(nf(n))$ for any $k \geq 1$.*

Proof. Let $k \geq 1$, $L \subseteq V^*$, and $w \in V^*$ for some alphabet V. We prefer to give the argument in the form of a recognizing Boolean function for the language $L \rightharpoondown_k$.

Algorithm 1.

```
function Rec_Right(w, L, k);
begin
i := 1;
while (i + k + 1 ≤ n − i − k)
 if (w[1..i + k] = w[n − k − i + 1..n]^R) and (w[1..n − i] ∈ L)
  then Rec_Right:=true; halt
  else i := i + 1;
 endif;
endwhile;
Rec_Right:=false;
end.
```

Since a similar recognizer can be designed for the language $L \rightharpoonup_k$, we are done. □

The following question naturally arises: Is the n factor needed in the above proposition? Clearly, a pre-processing phase computing in $\mathcal{O}(f(n))$ time a Boolean array a defined by $a[i] = (w[1..i] \in L)$, for any language L in a class of languages $\mathcal{F}$, would imply that the n factor is not needed for the class $\mathcal{F}$. This holds for the classes of regular and context-free languages.

Proposition 2. 1. *The membership problem for $L \rightarrow_k$ is decidable in linear time for any $k \geq 1$, provided that L is a regular language accepted by a given deterministic finite automaton.*

2. *The membership problem for $L \rightarrow_k$ is decidable in cubic time for any $k \geq 1$, provided that L is a context-free language generated by a given context-free grammar.*

Proof. 1. Assume that the language L is accepted by the finite deterministic automaton $A = (Q, V, \delta, q_0, F)$, with $Q = \{0, 1, \ldots, p\}$, and w is a word of length n over V. The pre-processing phase can be designed as follows:

Algorithm 2.

```
begin
m[0] := 0;
for i = 1 to n
  m[i] := δ(a[i − 1], w[i]);
  a[i] := (m[i] ∈ F);
endfor
end.
```

It is easy to note that we store in $m[i]$ the state $\delta(0, w[1..i])$ for all $1 \leq i \leq n$. Then $a[i]$ is true iff $m[i]$ is a final state.

2. Assume now that the language L is generated by the grammar $G = (N, V, S, P)$ in the Chomsky normal form. Also let w be a word of length n over V. The CYK algorithm (see [7,12]) produces in cubic time a two-dimensional array m defined by $m[i][j] = \{A \in N \mid A \longrightarrow^* w[i..j]\}$ for any $1 \leq i \leq j \leq n$. Now it suffices to compute a by $a[i] = (S \in m[1..i])$ for every $1 \leq i \leq n$. Clearly, this pre-processing phase takes $\mathcal{O}(n^3)$ running time. □

The problem of whether or not the n factor is needed for other classes of languages is still open. Of course, another structure of the function Rec_Right in Algorithm 1 might lead to other classes for which the n factor is not needed.

3.2. The case of iterated hairpin completion

We start showing that the class of polynomially recognizable languages **P** under iterated hairpin completion. More precisely,

Proposition 3. *For every $k \geq 1$ and every language L recognizable in $\mathcal{O}(f(n))$ time, the iterated k-hairpin completion of L is recognizable in $\mathcal{O}(\max(n^2 f(n), n^3))$ time.*

Proof. Let w be a word of length n. Function $Rec^*(w, L, k)$ decides whether or not $w \in L(\rightarrow_k)^*$. The algorithm computes a two-dimensional array m as follows: $m[i][j] = 1$ iff $w[i..j] \in L(\rightarrow_k)^*$, for every $1 \leq i < j \leq n$ such that $j - i \geq 2k$. Values of m are computed recursively: we start by setting $m[i][j] = 1$ for all the words $w[i..j]$, with $j - i \geq 2k$, that are in L. Then, we set $m[i][j] = 1$ iff $w[i..j]$ can be obtained by the k-hairpin completion of a word that is already known to be in $L(\rightarrow_k)^*$. It is clear that $m[1][n] = 1$ is equivalent to $w \in L(\rightarrow_k)^*$.

Algorithm 3.

```
function Rec*(w, L, k);
begin
if n ≤ 2k + 2 then Rec* := (w ∈ L);
endif;
for i = 1 to n − 2k
  for j = i + 2k to n
```

```
   if w[i, j] ∈ L then m[i][j] := 1
   endif
  endfor
endfor
for l = 2k + 3 to n
  for i = 1 to n − l + 1
   j := i + l − 1;
   p := 0;
   for t = i to i + [(l−1)/2] − 1
    if w[t] = w[j − t + i] then p := p + 1
     else exitfor
    endif
   endfor
   if p ≥ k + 1 then
    for t = 1 to p − k
     if m[i][j − t] = 1 or m[i + t][j] = 1 then m[i][j] := 1
     endif
    endfor
   endif
  endfor
endfor
Rec* := (m[1][n] = 1);
end.
```

Observe that the first cycle requires $\mathcal{O}(n^2 f(n))$ time, while the second cycle requires $\mathcal{O}(n^3)$ time. In conclusion, the complexity of this algorithm is $\mathcal{O}(\max(n^2 f(n), n^3))$. □

Can we do it better? We present an algorithm which runs faster for classes of languages recognizable in sublinear time.

Proposition 4. *For every $k \geq 1$ and every language L recognizable in $\mathcal{O}(f(n))$ time, the iterated k-hairpin completion of L is recognizable in $\mathcal{O}(n^2 f(n))$ time.*

Proof. Let w be a given word of length n. First we compute an $n \times n$ matrix P defined by

$$P[i][j] = \begin{cases} \max(\{t \mid w[i..i+t-1] = \overline{w[j-t+1..j]}^R\} \cup \{0\}), & j - i \geq 2k \\ 0, \quad \text{otherwise.} \end{cases}$$

This matrix can be easily computed in time $\mathcal{O}(n^2)$ as shown below.

Algorithm 4.

```
procedure Compute_matrix (P, w);
begin
for p = 2 to n − 2k + 1
  i := p − 1;
  j := p + 2k − 1;
  while (i ≥ 1)&(j ≤ n)
   if w[i] = w[j] then P[i][j] := P[i + 1][j − 1] + 1
   endif;
   i := i − 1;
   j := j + 1;
  endwhile;
```

```
 i := p − 1;
 j := p + 2k;
 while (i ≥ 1)&(j ≤ n)
  if w[i] = w[j] then P[i][j] := P[i + 1][j − 1] + 1
  endif;
  i := i − 1;
  j := j + 1;
 endwhile;
endfor
```

The construction is based on the relation

$$P[i][j] = \begin{cases} P[i+1][j-1]+1, & \text{if } w[i] = \overline{w[j]} \\ 0, & \text{if } w[i] \neq \overline{w[j]}. \end{cases}$$

Second, we compute two vectors of size n, *right* and *left*, such that at any moment of the execution of the algorithm the following two conditions are satisfied:

- $right[i]$ equals the greatest p, found by the algorithm until that moment, such that $w[i..p] \in L(\rightarrow_k)^*$,
- $left[j]$ equals the least p, found by the algorithm until that moment, such that $w[p..j] \in L(\rightarrow_k)^*$.

Initially, we have $left[j] = i$ and $right[i] = j$, for all i, j such that $w[i..j] \in L$; $left[j] = 0$ and $right[i] = n + 1$, otherwise. Indeed, these two instructions are sufficient, since $w[i..j]$ is obtained by hairpin completion iff it can be obtained by hairpin completion from its longest prefix or suffix, obtained by hairpin completion as well.

Now, the new algorithm for testing the membership problem becomes

Algorithm 5.

```
function Rec_New*(w, L, k);
begin
if n ≤ 2k + 2 then Rec_New* := (w ∈ L);
endif;
for i = 1 to n − 2k
 for j = i + 2k to n
  if w[i..j] ∈ L then m[i][j] := 1
  endif
 endfor
endfor
Compute_matrix(P, w);
for l = 2k + 3 to n
 for i = 1 to n − l + 1
  j := i + l − 1;
  if (j > right[i] ≥ j − P[i][j] + k) then
   m[i][j] = 1;
   left[j] = i;
   right[i] = j;
  endif;
  if (i < left[j] ≤ i + P[i][j] − k) then
   m[i][j] = 1;
   left[j] = i;
   right[i] = j;
  endif;
 endfor
endfor
Rec_New* := (m[1][n] = 1);
end.
```

Now it is plain that the first cycle is the most time-consuming part of the algorithm as it can be accomplished in $\mathcal{O}(n^2 f(n))$ time. □

We have a similar problem to that in the previous section: Is the n^2 factor always needed? Again, it is not needed for context-free languages. However, we were not able to find a linear algorithm for regular languages, but a quadratic one.

Proposition 5. 1. *The iterated hairpin completion of a context-free language is recognizable in cubic time.*
2. *The iterated hairpin completion of a regular language is recognizable in quadratic time.*

Proof. 1. We observe that we can use the CYK algorithm to check whether $w[i..j]$ is in L for all $1 \leq i \leq j \leq n$ provided that L is context-free. This can replace the first cycle of the algorithm above; hence the overall complexity of the algorithm for context-free languages becomes $\mathcal{O}(n^3)$.

2. It is rather straightforward to check in quadratic time whether $w[i..j]$ is in L for all $1 \leq i \leq j \leq n$ provided that L is regular. □

4. Hairpin completion distance

The *hairpin completion distance* between two words x and y is defined as the minimal number of hairpin completions which can be applied either to x in order to obtain y or to y in order to obtain x. If none of them can be obtained from the other by iterated hairpin completion, then the distance is ∞. Formally, the k-hairpin completion distance between x and y, denoted by $HCD_k(x, y)$, is defined by

$$HCD_k(x, y) = \begin{cases} \min\{p \mid x \in y(\rightarrow_k)^p \text{ or } y \in x(\rightarrow_k)^p\}, \\ \infty, \quad \text{if neither } x \in y(\rightarrow_k)^* \text{ nor } y \in x(\rightarrow_k)^*. \end{cases}$$

We start our investigation of this measure by showing that it is not trivial, that is for every naturals k, n there exist words x and y such that $HCD_k(x, y) \geq n$. It suffices to take $x = a^{k+1}b\overline{a^k}$ and $y = a^{k+1}b\overline{a^{k+n}}$. Clearly, $HCD_k(x, y) = n$; hence the hairpin completion distance is not only non-trivial but also connected. It is obvious that given k, n and a word x there does not necessarily exist y such that $HCD_k(x, y) = n$. However, one can decide whether this is the case. We prove a more general result, namely:

Proposition 6. *Given k and x one can decide in $\mathcal{O}(|x|)$ time whether or not for every n there exists y_n such that $HCD_k(x, y_n) = n$.*

Proof. The key point of the proof is the following fact.

Fact. *Let $k \geq 1$ and $x \in V^+$ for some alphabet V. For every $n \geq 2$ there does exist y_n such that $HCD_k(x, y_n) = n$ if and only if there exists $z \in (x \rightarrow_k) \setminus \{x\}$ with $z \rightarrow_k \neq \{z\}$.*

Proof of the Fact. It is clear that if $HCD_k(x, w) = 2$, for some $w \in V^+$, then there is $z \in ((x \rightarrow_k) \setminus \{x\})$ such that $w \in ((z \rightarrow_k) \setminus \{z\})$.

Conversely, assume that $z \rightarrow_k \neq \{z\}$ for some $z \in ((x \rightarrow_k) \setminus \{x\})$. It immediately follows that there exist y_1, y_2 such that $HCD_k(x, y_i) = i$, $i = 1, 2$. Let $z = \alpha\beta\gamma\overline{\beta^R\alpha^R}$ such that $x = \alpha\beta\gamma\overline{\beta^R}$ with $\alpha, \gamma \in V^+$ and $|\beta| = k$ (the case $x = \beta\gamma\overline{\beta^R\alpha^R}$ may be treated analogously). Since $z \rightarrow_k \neq \{z\}$, there are $u_1, u_2, v, t_1, t_2 \in V^+$ such that the following conditions are satisfied:

(i) $\alpha\beta = u_1vt_1 = vu_2t_2$, $|v| = k$, $|u_1| = |u_2|$.
(ii) Either $u_1vt_1\gamma\overline{t_2^Ru_2^Rv^Ru_1^R} \in ((z \rightarrow_k) \setminus \{z\})$ or $u_1vu_2t_2\gamma\overline{t_1^Rv^Ru_1^R} \in ((z \rightarrow_k) \setminus \{z\})$.

We consider the former case, $u_1vt_1\gamma\overline{t_2^Ru_2^Rv^Ru_1^R} \in ((z \rightarrow_k) \setminus \{z\})$ (a similar reasoning is valid for the latter case). It follows that $u_1v = vu_2$, which has the solutions $u_1 = \delta\eta$, $v = (\delta\eta)^p\delta$, $u_2 = \eta\delta$ for some $\delta, \eta \in V^*$ (at least one of them is nonempty) and $p \geq 0$. Consequently,

$$w = \delta\eta(\delta\eta)^p\delta t_1\gamma\overline{t_2^R\delta^R\eta^R\delta^R(\eta^R\delta^R)^p\eta^R\delta^R} \in ((z \rightarrow_k) \setminus \{z\}).$$

From the form of w one can easily infer that for every $n \geq 3$, there exists $y_n \in w(\rightarrow_k)^+$ such that $HCD_k(x, y_n) = n$, which concludes the proof of the fact. □

The proof of the proposition is complete as soon as we show that the test mentioned in the fact can be done in $\mathcal{O}(|x|)$ time. Clearly, the test can be algorithmically done as $x \rightarrow_k$ is a finite and effective set, which implies that all sets $z \rightarrow_k$, $z \in ((x \rightarrow_k) \setminus \{x\})$, are also finite and effective. However, this does not assure the time complexity claimed by the proposition. By a similar reasoning to that used in the proof of the above fact, it suffices to consider the following algorithm:

Algorithm 6.

function *Check_Fact*(x);
begin
compute $z = \alpha\beta\gamma\overline{\beta^R\alpha^r}$ *such that* $x = \alpha\beta\gamma\overline{\beta^R}$ *and* $|\beta\gamma|_\beta = 1$;
if $z \neq x$ then
  if (*the complement of the mirror image of the suffix of length* k *of* z *is a subword of the prefix of length* $|z| - k - 1$ *of* z) then *Check_Fact*:= true; halt
  endif
  if (*the complement of the mirror image of the prefix of length* k *of* z *is a subword of the suffix of length* $|z| - k - 1$ *of* z) then *Check_Fact*:= true; halt
  endif
endif
compute $z = \alpha\beta\gamma\overline{\beta^R\alpha^r}$ *such that* $x = \beta\gamma\overline{\beta^R\alpha^R}$ *and* $|\gamma\overline{\beta^R}|_{\overline{\beta^R}} = 1$;
if$z \neq x$ then
  if (*the complement of the mirror image of the suffix of length* k *of* z *is a subword of the prefix of length* $|z| - k - 1$ *of*z) then *Check_Fact*:= true; halt
  endif
  if (*the complement of the mirror image of the prefix of length* k *of* z *is a subword of the suffix of length* $|z| - k - 1$ *of* z) then *Check_Fact*:= true; halt
  endif
endif
Check_Fact:= false
end.

In the above algorithm for two words $u, v \in V^+$ we used the notation $|u|_v = \text{card}\{t \mid u = tvt', t \in V^*, t' \in V^+\}$. The time complexity of this algorithm is linear with respect to the length of x, which completes the proof. □

Algorithm 5 presented in the previous section can be easily modified to compute the hairpin distance between two words. We observe that if y can be obtained by applying n hairpin completions to x, then it has a subsequence z, such that y can be obtained from z by applying exactly once the hairpin completion, and z is obtained from x after applying exactly $n - 1$ hairpin completions. Moreover, if n is the hairpin distance between x and y, then z cannot be obtained by applying fewer than $n - 1$ hairpin completions to x. Using this observation, one can easily compute the distance between two words x and y by dynamic programming.

In what follows we assume that $|x| < |y| = n$. If the two words are of the same length, then the distance between them is either 0, if they coincide, or ∞ otherwise. If $|y| < |x|$, then we can interchange the two words. The data structures that we use are the following matrices:

- H, where $H[i][j]$ is the minimal number of hairpin completion applied to x in order to get $y[i..j]$;
- P described in the previous section.

The algorithm below computes the distance between x and y under the aforementioned assumptions. It is not hard to see that the algorithm runs in $\mathcal{O}(n^3)$, and that it can be modified to obtain a description of the hairpin completions that were applied to obtain y from x. This can be done by using an additional data structure that memorizes, for every i and j, the position of the hairpin structure in the current word which produces $y[i..j]$ by completion.

Algorithm 7.

```
function Hairpin_completion_distance(x, y);
begin
for i = 1 to n
 for j = 1 to n
  if y[i..j] = x then H[i][j] := 0 else H[i][j] := ∞;
  endif
 endfor
endfor
Compute_matrix (P, y);
for l = 2k + 3 to n
 for i = 1 to n − l + 1
  j := i + l − 1;
  for t = 1 to P[i][j] − k
   m := min(H[i][j − t] + 1, H[i + t][j] + 1);
   if (m < H[i][j]) then H[i][j] := m;
   endif;
  endfor
 endfor
endfor;
Hairpin_completion_distance := H[1][n];
end.
```

Therefore, we have:

Proposition 7. *The hairpin completion distance between two words x and y can be computed in $\mathcal{O}(\max(|x|, |y|)^3)$.*

It is worth mentioning that this algorithm may be used for checking the inclusion problem: Given k and two words x and y, does $y(\rightarrow_k)^* \subseteq x(\rightarrow_k)^*$ hold? It follows that the equivalence problem is also decidable: Given k and two words, does $y(\rightarrow_k)^* = x(\rightarrow_k)^*$ hold? However, the equivalence problem can be solved more efficiently, namely to check whether or not $x = y$.

5. Final remarks

We briefly discuss here a very few problems remained unsolved which seem attractive to us:

(1) As shown above, the inclusion problem: "Given k and two words x and y, does $y(\rightarrow_k)^* \subseteq x(\rightarrow_k)^*$ hold?" is decidable. A bit more general problem, namely: "Given positive integers k, p and two words x, y, does $y(\rightarrow_k)^* \subseteq x(\rightarrow_p)^*$ hold?" naturally arises. Does it remain decidable? This extension can be considered for the equivalence problem as well.
(2) For every word x and integer $k \geq 1$, the language $x(\rightarrow_k)^*$ is recognizable in quadratic time. Is this language always regular or context-free? If not, is it possible to decide this?
(3) Given k and two words x, y, is it decidable whether or not the intersection $x(\rightarrow_k)^* \cap y(\rightarrow_k)^*$ is empty? In the affirmative, what is the complexity of the algorithm?

These are only a very few problems that we have focused on. Of course, the reader may easily identify many other interesting and open problems.

Acknowledgements

The first author's work was partially supported by the Research grant ET75/2005 of the Romanian Authority for Scientific Research. The third author's work was partly supported by the Alexander von Humboldt Foundation.

References

[1] P. Bottoni, A. Labella, V. Manca, V. Mitrana, Superposition based on Watson–Crick-like complementarity, Theory of Computing Systems 39 (2006) 503–524.

[2] J. Castellanos, V. Mitrana, Some remarks on hairpin and loop languages, in: M. Ito, G. Păun, S. Yu (Eds.), Words, Semigroups, and Translations, World Scientific, Singapore, 2001, pp. 47–59.

[3] D. Cheptea, C. Martin-Vide, V. Mitrana, A new operation on words suggested by DNA biochemistry: Hairpin completion, in: Proc. Transgressive Computing, 2006, pp. 216–228.

[4] R. Deaton, R. Murphy, M. Garzon, D.R. Franceschetti, S.E. Stevens, Good encodings for DNA-based solutions to combinatorial problems, in: L.F. Landweber, E. Baum (Eds.), Proc. of DNA-based computers II, in: DIMACS Series, vol. 44, 1998, pp. 247–258.

[5] M. Garzon, R. Deaton, P. Neathery, R.C. Murphy, D.R. Franceschetti, E. Stevens, On the encoding problem for DNA computing, in: The Third DIMACS Workshop on DNA-Based Computing, Univ. of Pennsylvania, 1997, pp. 230–237.

[6] M. Garzon, R. Deaton, L.F. Nino, S.E. Stevens Jr., M. Wittner, Genome encoding for DNA computing, in: Proc. Third Genetic Programming Conference, Madison, MI, 1998, pp. 684–690.

[7] T. Kasami, An efficient recognition and syntax-analysis algorithm for context-free languages, Air Force Cambridge Research Laboratory, Bedford Mass, 1965.

[8] L. Kari, S. Konstantinidis, P. Sosik, G. Thierrin, On hairpin-free words and languages, in: C. De Felice, A. Restivo (Eds.), Proc. Developments in Language Theory 2005, in: LNCS, vol. 3572, Springer-Verlag, Berlin, 2005, pp. 296–307.

[9] Gh. Păun, G. Rozenberg, T. Yokomori, Hairpin languages, International Journal of Foundations of Computer Science 12 (2001) 837–847.

[10] G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, 3 volumes, Springer-Verlag, Berlin, Heidelberg, 1997.

[11] K. Sakamoto, H. Gouzu, K. Komiya, D. Kiga, S. Yokoyama, T. Yokomori, M. Hagiya, Molecular computation by DNA hairpin formation, Science 288 (2000) 1223–1226.

[12] D.H. Younger, Recognition and parsing of context-free languages in time n^3, Information and Control 10 (1967) 189–208.